Can Fireflies Gossip and Flock?: The possibility of combining well-know bio-inspired algorithms to manage multiple global parameters in wireless sensor networks without centralised control.

Michael Breza, Julie McCann
Department of Computing, Imperial College, London email: [mjb04, jamm]@doc.ic.ac.uk

Abstract

We have found that the structure of many well-known emergent algorithms are very similar. In this paper we discuss the structure and its elements, how it relates to algorithms that show stable patterns, and we look at some of the algorithms that can be built with this structure.

1. Introduction

In this paper we use algorithms that show emergent behaviour to manage multiple global system parameters in wireless sensor networks.

Wireless sensor networks (WSN) are networks of small resource constrained devices which sense the environment and report the results via wireless networks. They allow spacial/temporal measurements of phenomenon previously impossible to analyse [8]. One of the current challenges in the WSN field is the developemnt of management systems which allow WSN to be easily deployed in various application domains [13, 15]. Different WSN application domains often have different management requirements.

A definition of an emergent algorithm is given in [7] which states that an emergent algorithm produces predictable, or stable global effects with respect to one or more system parameters, by communicating with only immediate neighbours and in the absence of global control or information. Our method of engineering emergent algorithms is to base them on a simple algorithmic structure which is common to many algorithms which show emergent behaviour.

Many bio-inspired algorithms exist that use simple rules and local information to create a global consensus of a single parameter, such as clock time in the flashing firefly algorithm [16], or location in the slime algorithm [24].

In this paper we identify a control loop as the core structure of several bio-inspired algorithms. Two ways of combining these loops are identified, serially, and nested. These methods of combination can be used to combine algorithms to allow the control of multiple parameters (one per algorithm) on a global level without the use of a centralised point of control.

2. Principles of collective animal behaviour

A paper called "The Principles of Collective Animal Behaviour" [20] provides a list of primitives which we use to understand algorithms that exploit emergent behaviour. The principles given are: Integrity and variability, positive feedback, negative feedback, response thresholds, leadership, redundancy, synchronisation, and selfishness.

Some of the principles are already in heavy use, such as positive/negative feedback and integrity(randomness). Others like response thresholds and redundancy are such intrinsic parts of distributed systems as to seem obvious and not worthy of note, but the recognition of them is nonetheless important to understand their potential role in the engineering of emergent systems.

Leadership is an architectural primitive. Its existence in biological systems, and its benefit to performance in distributed systems means that its further consideration is important. Synchronisation is interesting in that it gives us a possible primitive to use for the convergence of temporal phenomenon. Selfishness gives us a way to assess the value of a unit participating in a group, and possibly defines conditions for an algorithm that uses emergence.

The recognition of these principles can be used to understand the structure of algorithms that show emergent behaviour. We aim to try and use these primitives to build new algorithms out of the ones already understood, to control global system parameters through emergence

without using any central control.

3. A reduction of the principles, and their application to bio-inspired algorithms

We focus on four of the principles of collective behaviour, because they are fundamental to bio-inspired algorithms which show emergent behaviour. Of these four, two are fundamental to the structure of emergent algorithms; which we refer to as a control loop. The other two are types of behaviour that the algorithms can exhibit. The four principles we focus on are:

- Integrity and variability (randomisation).
- Response threshold.
- Positive feedback.
- Negative feedback.

The other principles can either be expressed by one of the principles above, are higher level concepts, or express constraints on emergent algorithms. Synchronisation is the equivalent of positive feedback with the adapted state variable being time. Inhibition in its simplest state is simply negative feedback. Leadership is a higher level architectural concept, and relates to the management architecture differences of purely decentralised systems with no leadership, and hybrid systems with local area leaders creating a smaller, higher level of leadership above that of the regular node.

Redundancy and Selfishness both suggest some possible constraints for adaptive algorithms. Redundancy may suggest a minimum population needed for a given algorithm to work. As an example, the emergent leader algorithm in [1] assumes large systems with a high level of redundancy in order to justify its use of non-determinism. ACO [5] does not have the same constraint, but the time it takes to find a solution is increased as the ant population decreases. The constraint suggested by selfishness is that the algorithm must provide something to the participant in order to justify involvement. For instance: in a Gossip based algorithm, each node is willing to participate and initiate contact with a random neighbour at a given period in time, and listen for code updates the remainder of time. A node's participation in the system means that it receives updates rapidly, while its overall energy expenditure is minimal. It could only listen, and never participate in the active communication side of the algorithm. That would mean that it would have to listen for longer, and thereby use more energy to receive updates.

loop
  Measure outside world
  if Response threshold is met or not met **then**
    Adjust state variable predictably or based on a probabilistic choice.
  else
    Adjust state variable, randomly move, or just repeat.
  end if
end loop

Figure 1. Control loop observed in algorithms displaying emergent behaviour

3.1. A common structure, the control loop.

The first two principles; response thresholds and randomisation, are combined into a structure that is common to all of the bio-inspired algorithms that we discuss in this paper. The way these principles are combined is surprisingly regular, and creates a structure which we refer to as a control loop, because it is similar to the control loop given in control theory. This loop is shown in figure 1

Response thresholds can be see as conditional execution. The if statement in figure 1 embodies the use of response thresholds. All of the algorithms we have seen start by sampling the environment or their state, and based on a threshold or condition, will change their own state in some way. This process forms a loop, and by constantly iterating this loop, the agent keeps some part of its state consistent with that of all of the other agents in the system. It creates this uniformity of state by using randomisation.

Randomisation is the second principle. These algorithms always contain some action or decision made in a random way. The most common is random movement of the agent if the threshold is not met. This often takes place in the else statement in figure 1. By moving randomly, and then sampling the environment, the agent develops a good average view of its local made up of random samples. When this involves agent to agent interaction, the result is an ever increasing random sample of the agent population. As the sample size gets larger, it gives a closer approximation of the actual global average.

Another common usage of randomisation is in the change made once the threshold is met. The change itself can be made probabilistically. By making probabilistic decisions, less than optimal symmetries can be broken up and the state space can be constantly explored

so that when system change means that a parameter is no longer sufficient, a better one can be found. An example of this is the situation in a synchronisation protocol where two stable groups synchronise out of phase with each other, preventing global synchronisation. In this case some random behaviour forcing one or more of the agents out of their stable group could help them to find the other group, and create a chain reaction leading to global synchronisation.

The other two principles, positive and negative feedback, occur when the agents decisions are made probabilistically, and the probabilities are changed dynamically . In positive feedback, the weights of the probabilistic choice are increased as their associated state is increased. The best example of this behaviour is seen with Ant Colony Optimisation [2], [5]. In ACO, the environment has a state value, called pheromone. To decide direction of movement, the agents measure the pheromone levels around them. A path with higher pheromone has a higher probability of being travelled, and having its pheromone level increased. This then increases the probability that another agent will choose the same direction, creating a positive feedback loop. Negative feedback is a gradual decreasing of the associated state value. Over time, the pheromone evaporates, reducing the probability that the direction will be chosen. The feedbacks can be used together to create a stable system.

We looked at the control loops of seven well known algorithms which show emergent behaviour: fireflies [3] [16] [22]; flocking [18] [12] [23];heatbugs [27]; moths [28]; termites [17] [25]; slime [24]; and gossip [9] [4]. The algorethms that were chosen come from the model library of the netlogo simulator [21]. These models were originally written in the netlogo language [26], a high level logo like language that makes the algorithms very easy to read and analyse.

We will look at four ways the control loops can be varied. These include: the number of control loops central to the algorithm (or states the algorithm can be in), if loops are nested, if the agents move, and the type of decision making process used to determine the threshold.

Table 1 summarises our comparison of algorithms that show emergent behaviour. We can see from a comparison of firefly and flocking that there are two forms of the control loop, and basic one and a nested one. Termites gives an example of the basic control loop being serially combined so that the same random movement can produce different results based on internal state (presence or absence of a wood-chip). In the next section we will look further into the combination of these algorithms.

Much of the flexibility of the control loop structure comes from its ability to accommodate different decision making processes. Our basic example of fireflies shows simple fixed decisions made from the results of the environmental measurements. Some randomisation can be added into the decision making process by using fixed probabilities to help break up non-optimal symmetries, as with heatbugs. The chance to make a poor short term choice of position can result in the exploration of more of the available space, and lead to the discovery of a better long term choice. Finally, the probabilities themselves can be changed by the decision making process giving us feedback loops as in the slime algorithm.

Lastly, the importance of some form of random movement of either the data or of the agent can be seen in the fact that it is found in every algorithm. This suggests that one of the central mechanisms of algorithms that show emergent behaviour is the creation of a global average by having each agent take a random sample of its neighbour population. Movement means that although the sample is taken only from the agents local neighbours, those neighbours will change, so that over time the random sample size will increase, and eventually converge with that of the global average. This phenomenon is well understood in the area of epidemic algorithms (gossip) [6] [9]. Work already exists on gossip based systems for the aggregation of distributed data [10] [11].

The next issue is to attempt to use these control loops to allow several system parameters to be controlled at the same time. In order to test this idea in the next section we use the two observed forms of emergent control loop combination, serial and nested and create an algorithm using each method.

4. Combination of emergent control loops

From the discussion above, we find two ways to combine control loops. The first is a serial combination. This method can be seen in the interactions of the multiple states of the Termites algorithm. This method is simple, and means that when the threshold of one control loop has been met, then the algorithm changes state to use another control loop responding to a different threshold. In the first threshold the termite looks for a wood chip, completing that the termite looks for a wood pile (the second threshold), then the termite looks for an empty spot in the wood pile (the third threshold). The Termites algorithm is circular, once the termite has completed all three thresholds, it drops its wood chip, leaves the woodpile, and returns to the first control loop, looking for a wood chip.

	Fireflies	Flocking	Heatbugs	Moths	Termites	Slime	Gossip
Number of States	1	1	1	1	3	1	2
Nested	no	yes	no	yes	no	no	no
Agent movement	yes	yes	yes	yes	yes	yes	no
Decision	fixed	fixed	probabilistic	?probabilistic?	fixed	feedback	fixed

Table 1. Summary of all algorithms looked at.

The second combination method is to nest the control loops. This method is seen in both the Flocking and Moth algorithms. In both cases, the meeting of one threshold then leads to another threshold. For the flocking, the first threshold is the discovery of a neighbour, the second is distance from the neighbour. If a neighbour is too close, then move away, else align movement with the neighbour. The Moth algorithm is similar, once the light has been found, an intensity thresholds determines if the light is approached or circled.

The next question is whether these methods of combination can be used to combine the emergent algorithms themselves to allow us to control multiple system parameters at the same time. To test this idea, we created an algorithm using each of our combinatorial methods. The firefly and flocking algorithms were combined serially to create fireflies that flock, synchronising both flash times and direction of movement. Gossip and firefly were nested so that fireflies can flash together at the same time and in the same colour. Below we describe each algorithm and show the pseudo code.

4.1. Flocking-Fireflies

In figure 2 we can see the pseudo code for the flocking-firefly algorithm. It is made up of the firefly algorithm immediately followed by the Flocking algorithm. In this case, the successful meeting of one threshold is not required to move to the next state (as in Termites), but both control loops share the initial search for neighbours. Both control loops are nested. In the case of flocking, the existence of neighbours is followed by the distance to the closest neighbour to determine how to change state. With firefly the presence of the threshold number of flashing neighbours causes the change of state (clock reset). This algorithm succeeds in combing both control loops, synchronising both flash times and motion direction.

```
loop
  Find flock-mates
  if Flock-mates are found then
    measure the number of flock-mates flashing.
    if (state == NOT flashing) AND (flock-mates
    flashing ≥ threshold) then
      reset clock to reset point. The clock is reset
      based on the randomly obtained neighbour set.
    else
      do nothing.
    end if
  end if
end loop
loop
  Find nearest neighbour
  if Nearest neighbour exists then
    if Nearest neighbour is not too close then
      align to move with nearest neighbour
    else if Nearest neighbour is too close then
      separate from nearest neighbour
    end if
  else
    move randomly
  end if
end loop
```

Figure 2. Pseudo code for the flocking-fireflies algorithm made by serially combining the flocking and the fireflies algorithm

4.2. Firefly-Gossip

The Firefly-Gossip protocol shown in figure 3 works by combining two emergent algorithms. The type of combination is nesting, and the Gossip algorithm is nested inside the Firefly algorithm. First we will begin

```
local_clock = 0
cycle_length = cycle_length
next_broadcast = random(0, cycle_length)
local_metadata = 0
same_count = 0
loop
  if local_clock = next_broadcast then
    transmit local_clock and local_metadata value
    local_clock = local_clock + 1
    restart at top of loop
  end if
  if clock ≤ cycle_length then
    listen
    if A message is overheard then
      adjust local_clock to average of local_clock
      and time in message
    end if
    if The message contains metadata > lo-
    cal_metadata then
      transmit metadata now
    else if the message contains metadata < lo-
    cal_metadata then
      transmit data now
    else if the message contains metadata == lo-
    cal_metadata AND time in message == lo-
    cal_clock then
      same_count = same_count + 1
    end if
  end if
  if same_count > same_threshold then
    next_broadcast = 0
  end if
  local_clock = local_clock + 1
end loop
```

Figure 3. Pseudo code for the Firefly-Gossip algorithm made from nesting a Gossip algorithm inside of a Firefly synchronisation algorithm.

with a description of the Firefly algorithm, then we will explain the nested Gossip algorithm.

The Firefly part of the algorithm is based on the use of a listening window. When a node starts, it listens for a neighbour broadcast. If it hears any neighbours, it changes its clock time to the average of the clock times advertised in the message packets of its neighbours. Eventually the listening time windows of all of the agents synchronises.

The Gossip part of the algorithm uses a polite gossip similar to Trickle [14]. During a listen period, each node randomly schedules a broadcast containing its clock time at time of broadcast (assuming a MAC level time-stamping as per Trickle), and the metadata of any data the node may have to propagate (in our case, node colour). If during the time period before the scheduled broadcast, the node hears another broadcast with the same clock time and the same metadata, then that node will suppress its transmission (be polite). If the agent hears some metadata that is newer than the metadata it has, then it immediately broadcasts its old metadata. The reason for this is that when the agents hear a broadcast containing old metadata, they immediately broadcast the new data. When one agent begins to broadcast data, the others stay quite and cancel their broadcasts.

The nesting of Firefly and Gossip is different to the nesting seen in Moths and Flocking. In Firefly-Gossip, the nesting is temporal, the gossip only happens during the time slot that Firefly is synchronising. In Moths and Flocking, one threshold must be satisfied before the next one will be evaluated. In the case of Firefly-Gossip, the threshold for Gossip can be seen as being in the listening period of Firefly.

Using this form of temporal nesting, it is possible to combine the capabilities of the Gossip and Firefly algorithms and synchronise time and other parameters. This algorithm could be used in a wireless sensor network to both synchronise the clocks of the sensors and disseminate control information, all from one algorithm.

5. Performance of the combined algorithms

Some analysis via simulation was done to determine if the combination of algorithms affected their performance. The main performance focused upon was time for the agents of the system to synchronise their clocks and achieve convergence of a give system parameter. Experiments were run for both Flocking-Fireflies and Firefly-Gossip. All of the simulations were written in netlogo, and run on a 64 CPU cluster computer.

The results shown are both the averages of the results of 100 experiment runs, and the standard deviation. We say that the results shown for a given plot point show

equivalent performance if the plot point of one algorithm falls within one standard deviation of the other.

5.1. Flocking-Fireflies Experiments

In Flocking-Fireflies the time for a given percentage of the population to have the same direction (flocking) was measured for the flocking algorithm on its own, and when combined with the firefly synchronisation algorithm. The time to synchronise clocks (and flashes) was also measured for both the firefly synchronisation algorithm on its own, and when combined with the flocking algorithm.

In figure 4 we see the average population of agents flocking over time. By 500 time units (clock ticks) both algorithms have 90% of the agents flocking in the same direction. The graph shows that the performance for flocking is unaffected by the addition of the firefly algorithm.

Figure 5 shows us a very different situation. The addition of the flocking behaviour to the firefly algorithm reduces the time to synchronise. This effect is especially pronounced at lower populations of agents. This result is not surprising because the result of the flocking algorithm is to drive the agents closer together and move in the same direction. The agents not flocking just move about randomly. When the agents group together, their local neighbour population increases, and they are able to share information much more rapidly and therefore synchronise in a shorter period of time. This explains why the synchronisation times are much closer at higher populations where the local neighbour population of an agent is very high regardless of whether it is flocking or not.

5.2. Firefly-Gossip Experiments

With Firefly-Gossip we measured and compare the time to synchronise and the convergence rate of data (time to disseminate data to all nodes). Synchronisation time of the firefly algorithm on its own and convergence time of the gossip algorithm on their own are compared to the respective performance for the Firefly-Gossip algorithm. The agents in these experiments do not move, and are organised in a grid topology. The communication range is fixed at 15 patches. A patch is a spatial unit in netlogo. Each agent occupies one patch.

The x axis of the graphs 6 and 7 plots the inter-node distance in patches. As the inter-node distance increases, the average number of neighbours a node can hear gets smaller. The inter-node distances of eight, nine and ten patches all have very similar average neighbour populations (8.10, 8.03, and 7.94 respectively) and the same median neighbour population of nine neighbours. These similarities mean that the population of the network becomes the deciding factor of convergence and synchronisation rates. Since the area the simulation was run in was fixed, this means that the population of nodes decreased as the inter-node distance increased. The average neighbour population for each node decreased as the inter-node distance increased between the distances of two and eight patches. As stated above, eight, nine and ten patches had very similar numbers of neighbours. The inter-node distances of eleven through fifteen also had similar average neighbour populations (4.6 to 4.4). So, in those two groups of inter-node distances, we choose to plot only the middle values of nine and thirteen.

Both graphs show very similar performance. The average time to converge for data (figure 6) is consistently greater for Gossip-Firefly than for gossip on its own. Both averages are still within one standard deviation, and the differences are very small. In the synchronisation results (figure 7) it can be seen that neither algorithm outperforms the other, evident from the fact that the averages are almost the same for inter-node distances two through nine. The differences are the same scale as those for data convergence, given that the range of the y axis in figure 7 is greater than that for 6.

5.3. Discussion of the Experiments

In all cases we found the performance of the combined algorithms the same as or better than the algorithm on its own. The time to synchronise of the Firefly-Flocking algorithm was better (shorter) than the firefly algorithm on its own. This is because the addition of the flocking algorithm increases the average neighbour population of any given agent.

These experiments show that the combination of certain algorithms with others may have a beneficial effect to the performance of one of the algorithms. By the same logic, there must be certain combinations of algorithms that can produce detrimental effects on performance. This line of enquiry is left for future work.

6. Related Work

The idea of creating gossip based applications by composing different gossip services is described in [19]. In this work, a group of gossip based services are suggested. These services include a peer-sampling service upon which all other services are built, and a broadcast service, group composition services, and distributed computation service. These services are composed serially in a component based fashion. The outputs of one service becomes the input of another. Our work

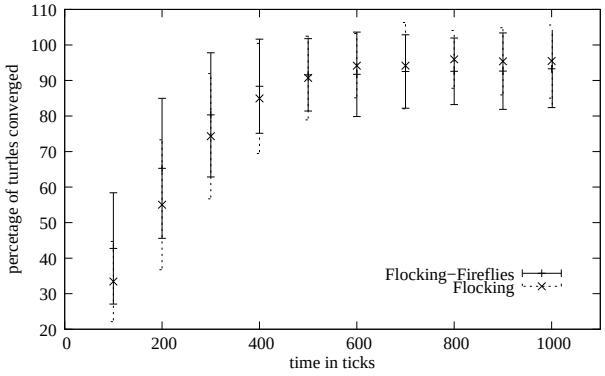

Figure 4. Average percentage of converged turtles over time. Note: the standard deviation is shown, which is why the y axis error bars maximum values sometimes go above 100%.

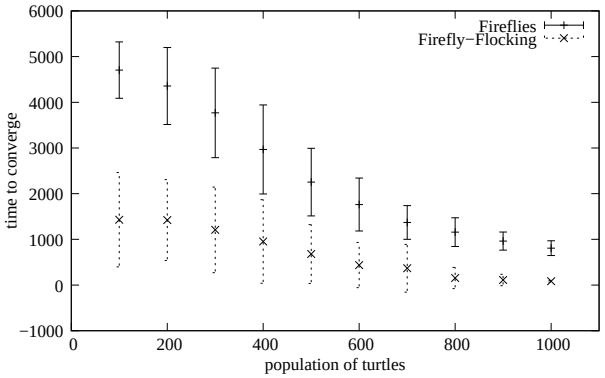

Figure 5. Time for turtles to synchronise for different populations of turtles.

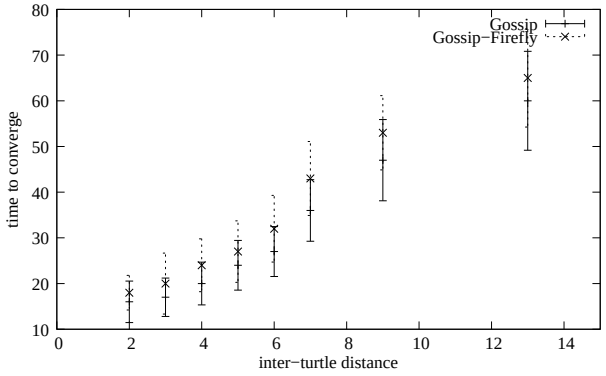

Figure 6. Time for turtle data to converge over different turtle densities. Note: In this case the turtle communication range was always 15.

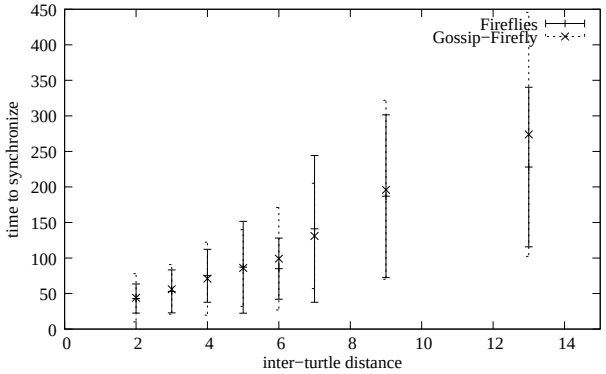

Figure 7. Time for turtles to synchronise time over different turtle densities. Note: In this case the turtle communication range was always 15.

deals with non-gossip based algorithms as well as gossip based algorithms, and includes a nested form of combination.

The combination of various bio-inspired algorithms is discussed in [] ADD CITATION. It views each algorithm as a search hueristic in a given search space. The heuristics are all common in that they have a method to create and refine a solution set. This work tries to optimize a given search by using methods of multiple algorithms over a single solution set. The goal is to produce a better solution set in a shorter period of time.

7. Conclusion

We have presented here a new approach to the use of algorithms exhibiting emergent behaviour to control multiple system parameters in a decentralised way. We start with the idea that many well know emergent algorithms are composed of basic principles, and that the principles are organised into a generic control loop. This control loop can be combined in a serial or nested fashion to allow the combination of emergent algorithms into larger ones that allow the decentralised control of multiple parameters. We presented two algorithms, one for each type of combination. For further work we will work with the Firefly-Gossip algorithm to adapt it to the application of decentralised wireless sensor network management.

References

[1] R. Anthony. An autonomic election algorithm based on emergence in natural systems. *Integrated computer-aided engineering*, 13(1):3 –22, 2006.

[2] C. Blum. Ant colony optimization: Introduction and recent trends. *Physics of Life Reviews*, 2(4):353–373, December 2005.

[3] J. Buck. Synchronous Rhythmic Flashing of Fireflies. II. *The Quarterly Review of Biology*, 63(3):265–289, 1988.

[4] A. Demers, D. Greene, C. Hauser, W. Irish, J. Larson, S. Shenker, H. Sturgis, D. Swinehart, and D. Terry. Epidemic algorithms for replicated database maintenance. *Proceedings of the sixth annual ACM Symposium on Principles of distributed computing*, pages 1–12, 1987.

[5] M. Dorigo and K. Socha. *Approximation Algorithms and Metaheuristics*, chapter An Introduction to Ant Colony Optimization. CRC Press, 2007.

[6] P. Eugster, R. Guerraoui, A. Kermarrec, and L. Massoulie. Epidemic information dissemination in distributed systems. *Computer*, 37(5):60–67, 2004.

[7] D. Fisher and H. Lipson. Emergent algorithms-a new method for enhancing survivability inunbounded systems. *System Sciences, 1999. HICSS-32. Proceedings of the 32nd Annual Hawaii International Conference on*, 1999.

[8] J. Hart and K. Martinez. Environmental Sensor Networks: A revolution in the earth system science? *Earth-Science Reviews*, 78:177–191, 2006.

[9] M. Jelasity. Engineering emergence through gossip. *Proc. AISB Convention, Joint Symposium on Socially Inspired Computing*, pages 123–126, 2005.

[10] M. Jelasity, A. Montresor, and O. Babaoglu. Gossip-based aggregation in large dynamic networks. *ACM Transactions on Computer Systems (TOCS)*, 23(3):219–252, 2005.

[11] D. Kempe, A. Dobra, and J. Gehrke. Gossip-based computation of aggregate information. *Foundations of Computer Science, 2003. Proceedings. 44th Annual IEEE Symposium on*, pages 482–491, 2003.

[12] J. Kennedy and R. Eberhart. Particle swarm optimization. *Neural Networks, 1995. Proceedings., IEEE International Conference on*, 4, 1995.

[13] W. L. Lee, A. Datta, and R. Cardell-Oliver. *Handbook of Mobile Ad Hoc and Pervasive Communications*, chapter Network Management in Wireless Sensor Networks. American Scientific Publishers, USA, 2006.

[14] P. Levis et al. *Trickle: A Self Regulating Algorithm for Code Propagation and Maintenance in Wireless Sensor Networks*. Computer Science Division, University of California, 2003.

[15] P. Marrón, A. Lachenmann, D. Minder, M. Gauger, O. Saukh, and K. Rothermel. Management and configuration issues for sensor networks. *International Journal of Network Management*, 15(4):235–253, 2005.

[16] R. Mirollo and S. Strogatz. Synchronization of Pulse-Coupled Biological Oscillators. *SIAM Journal on Applied Mathematics*, 50(6):1645–1662, 1990.

[17] M. Resnick. *Turtles, Termites, and Traffic Jams: Explorations in Massively Parallel Microworlds*. MIT Press, 1994.

[18] C. Reynolds. Flocks, herds and schools: A distributed behavioral model. *ACM SIGGRAPH Computer Graphics*, 21(4):25–34, 1987.

[19] É. Rivière, R. Baldoni, H. Li, and J. Pereira. Compositional gossip: a conceptual architecture for designing gossip-based applications. *ACM SIGOPS Operating Systems Review*, 2007.

[20] D. J. T. Sumpter. The principles of collective animal behaviour. *Philosophical transactions of the Royal Society of London*, 361:5–22, 2006.

[21] S. Tisue and U. Wilensky. NetLogo: A Simple Environment for Modeling Complexity. *International Conference on Complex Systems*, 2004.

[22] U. Wilensky. NetLogo Fireflies model. *Center for Connected Learning and Computer-Based Modeling, Northwestern University, Evanston, IL. http://ccl. northwestern. edu/netlogo/models/Fire flies*, 91.

[23] U. Wilensky. NetLogo Flocking model. *Web page http://ccl. northwestern. edu/netlogo/models/Flocking. Center for Connected Learning and Computer-Based Modeling, Northwestern University, Evanston, IL*, 1998.

[24] U. Wilensky. NetLogo Slime model. *Web page http://ccl. northwestern. edu/netlogo/models/Slime. Center for Connected Learning and Computer-Based Modeling, Northwestern University, Evanston, IL*, 1998.

[25] U. Wilensky. NetLogo Termites model. *Web page http://ccl. northwestern. edu/netlogo/models/Termites. Center for Connected Learning and Computer-Based Modeling, Northwestern University, Evanston, IL*, 1998.

[26] U. Wilensky. Modeling natures emergent patterns with multi-agent languages. *Proceedings of EuroLogo*, pages 1–6, 2002.

[27] U. Wilensky. NetLogo Heatbugs model. *Web page http://ccl. northwestern. edu/netlogo/models/Heatbugs. Center for Connected Learning and Computer-Based Modeling, Northwestern University, Evanston, IL*, 2004.

[28] U. Wilensky. NetLogo Moth model. *Web page http://ccl. northwestern. edu/netlogo/models/Moths. Center for Connected Learning and Computer-Based Modeling, Northwestern University, Evanston, IL*, 2005.